Un peu de calcul mental avec Python et C

Peter Philippe

30/11/2025

À l'heure de l'IA...

...où les capacités de raisonnement fondent comme neige au soleil, voici un petit programme permettant de récupérer une bribe de vivacité intellectuelle en effectuant des calculs élémentaires, tout en étant chronométré. Here we go !

Ce petit programme génère des équations linéaires fort simples avec les opérations de base que tout le monde connaît depuis l'école primaire. Le but est de résoudre 7 équations en calculant simplement la valeur de $X \in [1; 10]$ dans le temps imparti, fixé par défaut à 7 secondes par équation (ce qui fait, au pire, 49 secondes de stress).

Il ne sera pas nécessaire de se lancer dans le calcul du groupe de Galois en caractéristique 0 (quoique, non, je plaisante, c'est ici parfaitement inutile). Si l'on souhaite augmenter la difficulté, il suffit soit de réduire le temps de réponse, soit d'augmenter le nombre d'opérations ou bien d'agir sur ces deux paramètres simultanément.

Voici un exemple directement copié depuis le terminal (ne pratiquant plus ce genre d'exercice régulièrement, ce temps médiocre me sanctionne impitoyablement) :

```
--------------------------------------------------
              GÉNÉRATEUR D'ÉQUATIONS CHRONOMÉTRÉ
--------------------------------------------------
-> 7 équations linéaires à résoudre
-> 7 secondes par équation
-> X est un entier dans [1;10]
--------------------------------------------------

Question 1/7 :
Équation : X * 9 = 6 + 8 - 3 + 7 + 7 - 16
X = 1
Correct ! (5.1 s sur 7s)

Question 2/7 :
Équation : 3 * X = 4 + 9 + 5
X = 6
Correct ! (2.8 s sur 7s)

Question 3/7 :
Équation : X * 5 = 5 + 3 - 3 + 7 - 2 + 25
X = 7
Correct ! (5.3 s sur 7s)

Question 4/7 :
Équation : 5 * X = 7 + 8 + 2 - 7 + 7 + 3
X = 4
Correct ! (5.4 s sur 7s)

Question 5/7 :
Équation : X * 7 = 2 - 1 + 1 + 4 + 8
X = 2
Correct ! (4.9 s sur 7s)

Question 6/7 :
Équation : 7 * X = 9 - 6 + 10 - 8 + 3 - 1
X = 1
Correct ! (5.2 s sur 7s)

Question 7/7 :
Équation : X * 3 = 2 + 10 + 9 + 6
X = 9
Correct ! (3.9 s sur 7s)

--------------------------------------------------
              RÉSULTAT FINAL
--------------------------------------------------
Tu as calculé 7 bonnes réponses en 32.6s sur les 7 questions.
C'est parfait !
```

Voici donc les sources (Python puis C), ils sont également disponibles dans la partie publique mon compte Github (qui est un capharnaüm) :
`https://github.com/rayptor/linkedin`

```python
import random
import time

MIN_X = 1
MAX_X = 10
QUESTIONS = 7
TEMPS_LIMITE = 7

def generer_partie_droite(valeur_cible: int) -> str:
    nombre_depart: int = random.randint(MIN_X, MAX_X)
    expression_droite: str = str(nombre_depart)
    valeur_actuelle: int = nombre_depart
    operations_droite: int = random.randint(1, 4)

    for _ in range(operations_droite):
        operateur: str = random.choice(['+', '-'])
        nombre: int = random.randint(MIN_X, MAX_X)
        if operateur == '-' and valeur_actuelle - nombre < 1:
            operateur = '+'

        expression_droite += f" {operateur} {nombre}"
        valeur_actuelle += nombre if operateur == '+' else -nombre

    if (difference := valeur_cible - valeur_actuelle) != 0:
        expression_droite += f" {'+' if difference > 0 else '-'} {abs(difference)}"

    return expression_droite


def generer_partie_gauche() -> tuple[str, int]:
    x_coefficient: int = random.randint(MIN_X, MAX_X)
    x_solution: int = random.randint(MIN_X, MAX_X)
    valeur_cible: int = x_coefficient * x_solution
    expression_droite: str = generer_partie_droite(valeur_cible)
    equation: str = f"{x_coefficient} * X = {expression_droite}"

    return equation, x_solution


def generer_equation_equilibree() -> tuple[str, int]:
    x_solution: int = random.randint(MIN_X, MAX_X)
    nombre: int = random.randint(MIN_X+1, MAX_X)

    if random.choice([True, False]):
        expression_gauche: str = f"X * {nombre}"
        valeur_cible: int = x_solution * nombre
    else:
        if x_solution % nombre == 0:
            expression_gauche = f"X / {nombre}"
            valeur_cible = x_solution // nombre
        else:
            expression_gauche = f"X * {nombre}"
            valeur_cible = x_solution * nombre
```

```
    expression_droite: str = generer_partie_droite(valeur_cible)
    equation: str = f"{expression_gauche} = {expression_droite}"

    return equation, x_solution


def test() -> None:
    print("\n" + "-" * 50)
    print("\t\tGÉNÉRATEUR D'ÉQUATIONS CHRONOMÉTRÉ")
    print("-" * 50)
    print(f"-> {QUESTIONS} équations linéaires à résoudre")
    print(f"-> {TEMPS_LIMITE} secondes par équation")
    print("-> X est un entier dans [1;10]")
    print("-" * 50)

    score: int = 0
    limite: int = TEMPS_LIMITE
    nombre_questions: int = QUESTIONS
    temps_total: float = 0

    for i in range(1, nombre_questions + 1):
        generateur = random.choice([generer_equation_equilibree, generer_partie_gauche])
        equation: str
        solution: int
        equation, solution = generateur()
        print(f"\nQuestion {i}/{nombre_questions} :")
        print(f"Équation : {equation}")
        debut = time.time()
        try:
            reponse_utilisateur: str = input("X = ")
            temps: float = time.time() - debut
            est_correct: bool = False
            if reponse_utilisateur.isdigit():
                reponse: int = int(reponse_utilisateur)
                est_correct = (reponse == solution)

            if temps > limite:
                if est_correct:
                    print(f"Trop lent ({temps:.1f} s), mais la réponse est correcte !")
                    score += 1
                    temps_total += temps
                else:
                    print(f"Trop lent ({temps:.1f} s) !", end="")
                    print(f" (La bonne réponse était : X = {solution})")
            elif est_correct:
                print(f"Correct ! ({temps:.1f} s sur {limite}s)")
                score += 1
                temps_total += temps
            else:
                print(f"Faux ! ({temps:.1f} s)", end="")
                print(f" (La bonne réponse était : X = {solution})")

        except KeyboardInterrupt:
            print("\n\nQuiz interrompu !")
            break
        except ValueError:
            print("Erreur : Veuillez entrer un nombre entier valide.")
            continue
        except Exception as e:
            print(f"Erreur inattendue : {e}.")
            continue
```

```
    print("\n" + "-" * 50)
    print("\t\tRÉSULTAT FINAL")
    print("-" * 50)

    if score == nombre_questions:
        print(f"Tu as calculé {score} bonnes réponses en {temps_total:.1f}s "
              f"sur les {nombre_questions} questions.")
    else:
        print(f"Tu n'as calculé que {score} bonnes réponses "
              f"sur les {nombre_questions} questions.")

    match score:
        case n if n == nombre_questions:
            message = "C'est parfait !"
        case n if n == nombre_questions - 1:
            message = "C'est très bien !"
        case n if n >= nombre_questions // 2:
            message = "C'est moyen ! Tu peux mieux faire."
        case _:
            message = "C'est quoi ça ? Retourne donc t'entraîner !"

    print(message)

if __name__ == "__main__":
    test()
```

Et voici la version en langage C qui est forcément plus longue :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdckdint.h>
#include <time.h>

constexpr int X_MIN = 1;
constexpr int X_MAX = 10;
constexpr int NOMBRE_QUESTIONS = 7;
constexpr int TEMPS_LIMITE = 7;

constexpr size_t zoneTaille = 512;
constexpr size_t droiteTailleMax = zoneTaille - 80;

typedef struct {
    char equation[zoneTaille];
    int solution;
} Equation;

static inline int genererEntierAleatoire(int min, int max) {
    return min + rand() % (max - min + 1);
}

static inline bool genererBooleenAleatoire(void) {
    return (rand() & 1) != 0;
}

static inline void genererPartieDroite(
    int valeurCible,
    char* restrict buffer,
    size_t tailleTampon
```

```
) {
    if (buffer == nullptr || tailleTampon == 0) {
        return;
    }

    int valeurDepart = genererEntierAleatoire(X_MIN, X_MAX);
    int valeurActuelle = valeurDepart;
    int decalage = snprintf(buffer, tailleTampon, "%d", valeurDepart);

    if (decalage < 0 || (size_t)decalage >= tailleTampon) {
        return;
    }

    int nombreOperations = genererEntierAleatoire(1, 4);

    for (int i = 0; i < nombreOperations; ++i) {
        char operateur = genererBooleenAleatoire() ? '+' : '-';
        int nombre = genererEntierAleatoire(X_MIN, X_MAX);
        int nouvelleValeur;

        if (operateur == '-') {
            if (ckd_sub(&nouvelleValeur, valeurActuelle, nombre) || nouvelleValeur < 1) {
                operateur = '+';
                if (ckd_add(&nouvelleValeur, valeurActuelle, nombre))
                    break;
            }
        } else {
            if (ckd_add(&nouvelleValeur, valeurActuelle, nombre))
                break;
        }

        int zoneEcrite = snprintf(
            buffer + decalage,
            tailleTampon - (size_t)decalage,
            " %c %d",
            operateur,
            nombre
        );

        if (zoneEcrite < 0 || (size_t)decalage + (size_t)zoneEcrite >= tailleTampon)
            return;

        decalage += zoneEcrite;
        valeurActuelle = nouvelleValeur;
    }

    int difference = valeurCible - valeurActuelle;
    if (difference != 0) {
        char op = (difference > 0) ? '+' : '-';
        int absDiff = (difference > 0) ? difference : -difference;

        snprintf(
            buffer + decalage,
            tailleTampon - (size_t)decalage,
            " %c %d",
            op,
            absDiff
        );
    }
}

static void genererPartieGauche(Equation* restrict eq) {
```

```
    if (eq == nullptr)
        return;

    int coefficientX = genererEntierAleatoire(X_MIN, X_MAX);
    int solutionX = genererEntierAleatoire(X_MIN, X_MAX);
    int valeurCible;

    if (ckd_mul(&valeurCible, coefficientX, solutionX)) {
        coefficientX = X_MIN;
        solutionX = X_MIN;
        valeurCible = coefficientX * solutionX;
    }

    char expressionDroite[droiteTailleMax];
    genererPartieDroite(valeurCible, expressionDroite, sizeof(expressionDroite));

    int taillePartieDroite = snprintf(nullptr, 0, "%d * X = %s", coefficientX, expressionDroite);
    if (taillePartieDroite < 0 || (size_t)taillePartieDroite >= sizeof(eq->equation)) {
        snprintf(eq->equation, sizeof(eq->equation), "%d * X = 1", coefficientX);
        eq->solution = solutionX;
        return;
    }

    snprintf(
        eq->equation,
        sizeof(eq->equation),
        "%d * X = %s",
        coefficientX,
        expressionDroite
    );
    eq->solution = solutionX;
}

static void genererEquationEquilibree(Equation* restrict eq) {
    if (eq == nullptr)
        return;

    int solutionX = genererEntierAleatoire(X_MIN, X_MAX);
    int nombre = genererEntierAleatoire(X_MIN + 1, X_MAX);
    char expressionGauche[64];
    int valeurCible;

    if (genererBooleenAleatoire()) {
        snprintf(expressionGauche, sizeof(expressionGauche), "X * %d", nombre);
        if (ckd_mul(&valeurCible, solutionX, nombre)) {
            solutionX = X_MIN;
            nombre = X_MIN + 1;
            valeurCible = solutionX * nombre;
        }
    } else {
        if (solutionX % nombre == 0) {
            snprintf(expressionGauche, sizeof(expressionGauche), "X / %d", nombre);
            valeurCible = solutionX / nombre;
        } else {
            snprintf(expressionGauche, sizeof(expressionGauche), "X * %d", nombre);
            if (ckd_mul(&valeurCible, solutionX, nombre)) {
                solutionX = X_MIN;
                nombre = X_MIN + 1;
                valeurCible = solutionX * nombre;
            }
        }
    }
```

```
    char expressionDroite[droiteTailleMax];
    genererPartieDroite(valeurCible, expressionDroite, sizeof(expressionDroite));

    int taillePartieGauche = snprintf(nullptr, 0, "%s = %s", expressionGauche, expressionDroite);
    if (taillePartieGauche < 0 || (size_t)taillePartieGauche >= sizeof(eq->equation)) {
        snprintf(eq->equation, sizeof(eq->equation), "X * %d = 1", nombre);
        eq->solution = solutionX;
        return;
    }

    snprintf(
        eq->equation,
        sizeof(eq->equation),
        "%s = %s",
        expressionGauche,
        expressionDroite
    );
    eq->solution = solutionX;
}

static inline double calculerTemps(struct timespec debut, struct timespec fin) {
    return (double)(fin.tv_sec - debut.tv_sec) +
           (double)(fin.tv_nsec - debut.tv_nsec) / 1e9;
}

static void afficherSeparateur(void) {
    for (int i = 0; i < 50; ++i) {
        putchar('-');
    }
    putchar('\n');
}

[[nodiscard]] static bool lireReponse(
    char* restrict buffer,
    size_t taille,
    double* restrict tempsEcoule
) {
    struct timespec debut, fin;
    clock_gettime(CLOCK_MONOTONIC, &debut);

    printf("X = ");
    fflush(stdout);

    if (fgets(buffer, (int)taille, stdin) == nullptr)
        return false;

    clock_gettime(CLOCK_MONOTONIC, &fin);
    *tempsEcoule = calculerTemps(debut, fin);

    size_t len = strlen(buffer);
    if (len > 0 && buffer[len - 1] == '\n')
        buffer[len - 1] = '\0';

    return true;
}

static void test(void) {
    printf("\n");
    afficherSeparateur();
    printf("\t\tGÉNÉRATEUR D'ÉQUATIONS CHRONOMÉTRÉ\n");
    afficherSeparateur();
```

```
    printf("-> %d équations linéaires à résoudre\n", NOMBRE_QUESTIONS);
    printf("-> %d secondes par équation\n", TEMPS_LIMITE);
    printf("-> X est un entier dans [%d;%d]\n", X_MIN, X_MAX);
    afficherSeparateur();

    srand((unsigned int)time(nullptr));
    int score = 0;
    double tempsTotal = 0.0;
    char buffer[zoneTaille];

    for (int i = 1; i <= NOMBRE_QUESTIONS; ++i) {
        Equation eq = {0};
        if (genererBooleenAleatoire()) {
            genererEquationEquilibree(&eq);
        } else {
            genererPartieGauche(&eq);
        }

        printf("\nQuestion %d/%d :\n", i, NOMBRE_QUESTIONS);
        printf("Équation : %s\n", eq.equation);

        double tempsEcoule;
        if (!lireReponse(buffer, sizeof(buffer), &tempsEcoule)) {
            printf("\nErreur de lecture !\n");
            continue;
        }

        char* pointeurFin;
        long reponse = strtol(buffer, &pointeurFin, 10);

        bool estValide = !(pointeurFin == buffer || *pointeurFin != '\0');
        bool estCorrect = estValide && (reponse == eq.solution);

        if (!estValide) {
            printf("Faux ! (%.1f s)\n", tempsEcoule);
            printf("La bonne réponse était : X = %d\n", eq.solution);
        } else if (tempsEcoule > TEMPS_LIMITE) {
            if (estCorrect) {
                printf("Trop lent (%.1f s), mais la réponse est correcte !\n", tempsEcoule);
                ++score;
                tempsTotal += tempsEcoule;
            } else {
                printf("Trop lent (%.1f s) !\n", tempsEcoule);
                printf("La bonne réponse était : X = %d\n", eq.solution);
            }
        } else {
            if (estCorrect) {
                printf("Correct ! (%.1f s sur %ds)\n", tempsEcoule, TEMPS_LIMITE);
                ++score;
                tempsTotal += tempsEcoule;
            } else {
                printf("Faux ! (%.1f s)\n", tempsEcoule);
                printf("La bonne réponse était : X = %d\n", eq.solution);
            }
        }
    }

    printf("\n");
    afficherSeparateur();
    printf("\t\tRÉSULTAT FINAL\n");
    afficherSeparateur();
```

```
    if (score == NOMBRE_QUESTIONS) {
        printf("Tu as calculé %d bonnes réponses en %.1fs "
               "sur les %d questions.\n",
               score, tempsTotal, NOMBRE_QUESTIONS);
    } else {
        printf("Tu n'as calculé que %d bonnes réponses "
               "sur les %d questions.\n",
               score, NOMBRE_QUESTIONS);
    }

    const char* message;
    if (score == NOMBRE_QUESTIONS) {
        message = "C'est parfait !";
    } else if (score == NOMBRE_QUESTIONS - 1) {
        message = "C'est très bien !";
    } else if (score >= NOMBRE_QUESTIONS / 2) {
        message = "C'est moyen ! Tu peux mieux faire.";
    } else {
        message = "C'est quoi ça ? Retourne donc t'entraîner !";
    }

    printf("%s\n", message);
}

int main(void) {
    test();
    return EXIT_SUCCESS;
}
```